Geoscience Frontiers 11 (2020) 1859–1873

Contents lists available at ScienceDirect

Geoscience Frontiers

journal homepage: www.elsevier.com/locate/gsf

Research Paper

An efficient framework for ensemble of natural disaster simulations as a service

Ujjwal KC [a,*], Saurabh Garg [a], James Hilton [b]

[a] Discipline of ICT, University of Tasmania, Hobart, Australia
[b] Data61, CSIRO, Melbourne, Australia

ARTICLE INFO

Handling Editor: Sohini Ganguly

Keywords:
Wildfire prediction
Ensemble simulation
Cloud computing
Natural disaster models

ABSTRACT

Calculations of risk from natural disasters may require ensembles of hundreds of thousands of simulations to accurately quantify the complex relationships between the outcome of a disaster and its contributing factors. Such large ensembles cannot typically be run on a single computer due to the limited computational resources available. Cloud Computing offers an attractive alternative, with an almost unlimited capacity for computation, storage, and network bandwidth. However, there are no clear mechanisms that define how to implement these complex natural disaster ensembles on the Cloud with minimal time and resources. As such, this paper proposes a system framework with two phases of cost optimization to run the ensembles as a service over Cloud. The cost is minimized through efficient distribution of the simulations among the cost-efficient instances and intelligent choice of the instances based on pricing models. We validate the proposed framework using real Cloud environment with real wildfire ensemble scenarios under different user requirements. The experimental results give an edge to the proposed system over the bag-of-task type execution on the Clouds with less cost and better flexibility.

1. Introduction

Natural disasters cause a widespread loss of life and damage to infrastructure with associated economic losses. The advent of modern computational methods and hardware has allowed models to be developed to simulate and predict these complex phenomena. The models represent such complex phenomena that are contributed by a large number of factors. Due to this, the models usually have high computational requirements and are not feasible to run in an operational environment. Deriving accurate risk metrics from such models can require hundreds of thousands of possible scenarios, collectively referred to as an *ensemble*, to be run. However, even a single simulation is a complex calculation based on interrelationships between different parameters, and must also deal with geographical information data sets. Running ensembles on a single computer or a small cluster can result in bottlenecks due to data access and processing constraints. Thus, it may take several hours to days to fully cover the required perimeter space. Furthermore, in a real-time operational environment where ensemble simulations are being run to predict real wildfires, resource constraints from a limited computing pool may delay predictions required for operational management with unwanted consequences, for controlling fires effectively or timely evacuations from regions in danger.

Research carried out in recent years has put forward Cloud Computing frameworks as a possible solution to increase the efficiency of the prediction tools and make these services available to many users in a scalable way. Cloud Computing, which is based on principles of distributed computing, possesses the features of pooling, sharing, integrated computing technologies, and vast computer resources (Huang et al., 2018). Cloud infrastructure itself does not decrease the computation time for individual simulation in an ensemble. But, it provides a means to reduce the overall time of the ensemble as it allows elastic on-demand access to almost unlimited storage, network, and computational processing. However, this access to the Cloud resources must be coupled with an effective control mechanism in the system design to manage the resources and support the prediction models in optimal manners.

It is desirable to offer the functionality of ensemble simulations of disaster models as end services. However, the inherent nature of ensemble simulations can invite several challenges regarding the resource utilization, user requirements and cost incurred. For ease-of-use, there must also be an effective mechanism that can handle the ensemble simulations within the

* Corresponding author.
E-mail addresses: Ujjwal.KC@utas.edu.au (U. KC), Saurabh.Garg@utas.edu.au (S. Garg), James.Hilton@data61.csiro.au (J. Hilton).
Peer-review under responsibility of China University of Geosciences (Beijing).

https://doi.org/10.1016/j.gsf.2020.02.002
Received 24 June 2019; Received in revised form 1 December 2019; Accepted 8 February 2020
Available online 6 March 2020

Cloud environment without requiring frequent user interventions. Kalabokidis et al. (2013) initiated the use of Cloud Computing for fire simulation model, while Garg et al. (2018) provided a conceptual model to provide a scalable wildfire prediction over the Cloud environment. Garg et al. (2018) proposed sparkCloud service - a web-based Cloud platform system to demonstrate the elastic and scalable Cloud solution for wildfire prediction model based on user requests and deadline requirements. Ujjwal et al. (2019) proposed a conceptual solution framework to offer different disaster-related functionalities as a service over Cloud environment. However, no studies to date have clearly defined a mechanism for enabling the ensemble simulations of any natural disaster models as end services over the Cloud environment with optimized cost and resource utilization. Moreover, there are no specific studies that define how to enable ensemble simulations of natural disaster models over the Cloud foundation with minimal user interventions during the simulation run.

As such, this study puts forward a framework that helps in the realization of the ensemble of disaster simulations as end services over the Cloud environment. The proposed framework considers the user requirements and minimizes the cost of operation in two distinct phases. In the first phase, the possible incurred cost is minimized through efficient distribution of the simulations among cost-efficient workers while still complying to the user requirements. The second phase minimizes the cost of operation further by intelligently choosing the instances based on different pricing models - on-demand, reserved and spot. This study validates the working of the proposed system design by implementing the design with a wildfire prediction tool, Spark (Miller et al., 2015), in the Cloud environment. In the proposed system, end-users can ubiquitously access and use the ensemble services via a web interface using the internet with minimal cost. The contributions of this study are:

(1) A validated foundation system design (framework) to deploy the ensemble of wildfire simulations as end services over the Clouds considering the user requirements with minimal cost;
(2) A resource-centered scheduling mechanism that clusters the simulations in an ensemble based on the effective operation of the Cloud instances;
(3) A queueing theory-based Capacity Planner to save the time required for the creation of new cloud instances.

The rest of the paper is organized as follows: Section 2 highlights the related works, while Section 3 explains the associated challenges. Section 4 proposes a system design (framework) and Section 5 explains the evaluation of the proposed design in detail. Section 6 discusses the results collected from evaluation, while Section 7 concludes the paper with possible future extensions.

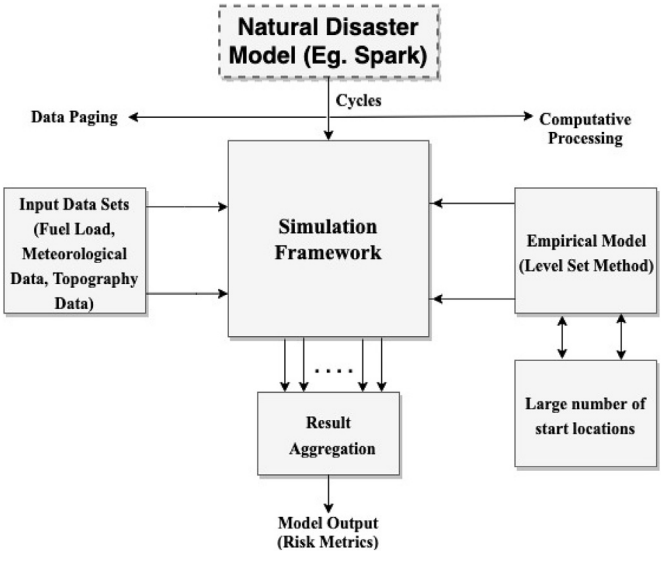


Fig. 1. An ensemble of a general disaster model.

2. Related works

Several studies have implemented geospatial models over the Cloud for different disaster management scenarios. Eriksson et al. (2011) developed a simulator in Amazon EC2 Clouds to understand the outbreak of pandemic influenza over a particular place. Wan et al. (2014) used Cloud infrastructure to classify the different occurrences of the flood into different levels based on severity and fatalities. The work done by Montgomery and Mundt (2010) processed different geospatial data sets using a Cloud environment to predict the changes of the natural resources. The climate engine (Huntington et al., 2017) was developed using Cloud infrastructure to forecast the weather through climatological calculations and related statistical analyses. Pajorová and Hluchỳ (2011) carried out complex Earth and astrophysics simulations using a Cloud environment. For wildfires, Kalabokidis et al. (2002) highlighted the need for quantitative indices of wildfire behavior and effects with spatial layers of meteorological, vegetative, topographic and socioeconomic information for a holistic fire risk assessment of hazards and vulnerability. Kalabokidis et al. (2013) proposed a web-based GIS platform called Virtual Fire using FARSITE (Farsite, 1998) over the Cloud that offers various fire management related services. The study accommodated the fire propagation simulation in Virtual Fire, but the end-users could not initiate fire behavior simulations for various technical and operational reasons. Kalabokidis et al. (2014) explained how wildfire risk and spread simulation services could be offered as Software as a Service (SaaS) over the Cloud environment with more flexibility. Garg et al. (2018) developed sparkCloud using Spark for wildfire prediction to demonstrate the capability of Cloud Computing to support different natural disaster models. However, the study focused on providing scalable solutions for running a wildfire propagation simulation within a Cloud environment based on user requirements without considering the ensemble with a large number of simulations.

Huang et al. (2013b) verified the capability of Cloud Computing to support ensemble simulations by deploying a complex dust forecasting model on an Amazon EC2 foundation with reduced cost when compared to using local resources. Li et al. (2017) described a Model as a Service (MaaS) framework to support ensemble simulations of different Geoscience models over the Cloud infrastructure. Moreover, a cyberinfrastructure based system developed by Behzad et al. (2011) detailed the implementation of ensemble simulation of groundwater system modeling over the Cloud environment provided by Microsoft Windows Azure Cloud Platform. These works have validated the readiness of Cloud infrastructure to support the complex ensemble simulations of different Geoscience models. However, fewer developments have been made to offer these models as end services to the users. Cost and resource optimization for ensemble simulations of natural disasters models over the Cloud environment have not, to our knowledge, been previously considered. Moreover, there are not any well-defined mechanisms to initiate and automate the multiple runs of simulations with minimal user interventions (a single user request) for an ensemble of disaster simulations.

The execution of simulations in an ensemble is conceptually similar to the execution of tasks in a bag-of-tasks application. These well-studied applications deal with a large number of independent tasks which can be executed in any order on any computational resource. However, for disaster models executing the simulations in variable batches, rather than as independent units, can significantly enhance the overall performance due to the large sizes of the input data sets, the sharing of intermediate data sets between different simulations and the specific geospatial requirements of the models. As highlighted in work by Thai et al. (2018), Cloud Computing has been widely adopted for bag-of-task applications due to flexibility in resource provisioning and on-demand pricing models. The optimization of the cost and the resource usage is focused on different perspectives of data centers and the users (Varghese and Buyya, 2018). There are different frameworks proposed in different works (Candeia et al., 2010; Bicer et al.,

2012; Duan and Prodan, 2014) where user-defined requirements, bandwidth and storage constraints and monetary cost are considered while executing the bag-of-task applications. These existing frameworks and mechanisms may not ensure reduced operational cost for ensembles of simulations as end services, and this is where the extension of the existing optimization schemes is required. Moreover, so far, the task clustering (creation of batches) has been done based on user requirements (time and budget) (Muthuvelu et al., 2010, 2013), bandwidth (Keat et al., 2006; Ang et al., 2009) and resource constraints (Muthuvelu et al., 2008). For the ensembles of disaster simulations, each simulation is both compute and data-intensive. Thus, the creation of batches of simulations based on the most effective operation regions of the machines for user requirements can be more efficient. The estimation of resources required to execute the requests can also be helpful. As such, this study considers the unique features of disaster models and simulations to schedule the simulations in an ensemble to offer such functionalities as end services with minimal cost and resources. This study also considers the capacity planning and different pricing models of Cloud instances.

3. Model and challenges

In this section, we first discuss the ensemble of a general disaster model with different components and phases of simulating the dynamics of the phenomenon over time. We then explain in detail the challenges associated with offering such ensembles of disaster simulations as end services.

3.1. An ensemble of natural disaster model

For disasters such as wildfires, the parameter space of factors affecting the fire can be mapped to possible outcomes allowing the detailed risk metrics to be calculated. These input factors can include parameters such as the starting location for the fire, the wind conditions, and the air temperature. The possible outcomes can be the total area burned and whether the fire impacts any areas with homes or infrastructure. The number of required simulations can scale exponentially with the number of input parameters. Natural disaster models such as Spark, usually consist of two distinct cycles - data paging and computative processing, to simulate the behavior of the disasters. An overview of an ensemble of a general disaster model is shown

```
  <input globalname="Curing/Layer default">100</input>
  <input globalname="Curing/Layer interpolation">1</input>
  <input globalname="Elevation/Layer default">0</input>
  <input globalname="Elevation/Layer projection WKT">EPSG:28355</input>
  <input globalname="Elevation/Layer source file">../../../../input/Terrain_inputs/Spark_DEM.tif</input>
  <input globalname="Elevation/Layer interpolation">1</input>
  <input globalname="Fire history/Layer default">0</input>
  <input globalname="Fire history/Layer projection WKT">EPSG:28355</input>
  <input globalname="Fire history/Layer source file">../../../../input/Terrain_inputs/FireHistorySept2017.tif</input>
  <input globalname="Fuel load/Layer interpolation">1</input>
  <input globalname="Classification/Layer projection WKT">EPSG:28355</input>
  <input globalname="Classification/Layer source file">../../../../input/Terrain_inputs/Fuel_TAS.tif</input>
  <input globalname="Gridded/Source directory">../../../../input/Meteorological_inputs</input>
  <input globalname="Gridded/Layer projection WKT">EPSG:4326</input>
  <input globalname="Gridded/Time conversion coefficient">1</input>
  <input globalname="Gridded/Wind/Layer direction source file"/>
  <input globalname="Gridded/Wind/Layer direction source filter">Wind_2MC_2018_32bit_Wind_Dir_SFC.nc</input>
  <input globalname="Gridded/Wind/Layer magnitude source file"/>
  <input globalname="Gridded/Wind/Layer magnitude source filter">*_WindMagKmh_*.nc</input>
  <input globalname="Gridded/Relative humidity/Layer source filter">*_RH_*.nc</input>
  <input globalname="Initialisation Python input file">../../../../input/FuelTypes.xml</input>
  <input globalname="Initialisation Python input file 2">../../../../input/Master_Spark_Lookup_v1_vesta.csv</input>
  <input globalname="Initialisation Python script">
"Define how far out radius should search in this case use 6km"
xy_range = [-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6]
range_len = len(xy_range)
centroid_easting = 538424.698422781
centroid_northing = 5246922.78512832
x_direction = int((seed - 1) / range_len)
y_direction = (seed - 1) - (range_len * x_direction)
nx_direction = int((seed - 1) / range_len)
ny_direction = (seed - 1) - (range_len * x_direction)
"Use a 1km grid of ignitions"
new_easting = centroid_easting + 1000 * xy_range[x_direction]
new_northing = centroid_northing + 1000 * xy_range[y_direction]
lat, long = utm.to_latlon(new_easting, new_northing, 55, 'G')
"******Spark inputs******"
lat = [lat]
long = [long]
radius = [200]
time = [0]</input>
  <input globalname="Start time">2018-12-31T13:00:00+11:00</input>
  <input globalname="End time"/>
  <input globalname="Number of simulations">10</input>
  <input globalname="Random seed">0</input>
```

Fig. 2. A sample XML configuration file with key configuration parameters.

in Fig. 1. In Data paging cycle, all the required input data sets are collected and fed into the simulation framework. During computative processing, empirical models are used to predict the progression of the disaster phenomenon over time. The key feature of an ensemble of disaster simulation is the requirement of hundreds to thousands of simulations to derive more accurate risk metrics. For operational management, any predictions about the outspread of the disaster can be significant in saving lives and physical properties.

3.2. Challenges

Predicting accurate risks of natural disasters using an ensemble has a principle challenge of managing the execution of a large number of simulations in time and resource-efficient manner. As such, all the challenges associated with developing different mechanisms to efficiently deploy the ensemble of disaster simulations as end services over a Cloud foundation, are described as follows.

3.2.1. Achieving ensemble of simulations over multiple cloud instances with minimal user intervention

While executing an ensemble of simulations over multiple Cloud instances, the scenarios for the ensemble have to be created through several simulations over a large number of start locations (Garg et al., 2018). These simulations have to be distributed over multiple instances. Running the simulations in batch mode can save time as a single data paging would work for all the simulations in the batch, but, the same is not true for computative processing. It can be optimal to divide the ensemble scenario into several groups of simulation as subjobs. These subjobs have to be independently assigned to the instances within the system. Moreover, the methods how the multiple outputs from each simulation are collected and stored during Result Aggregation and processed are equally important and challenging for better interpretation of the results (Ujjwal et al., 2019). Achieving all these requirements effortlessly with minimal user intervention can be a big challenge.

3.2.2. Supporting computational complexity of ensemble simulations over the cloud environments with optimal resource utilization

With the features of almost unlimited compute, network, and storage, Cloud Computing can support the computational complexities of ensemble simulations. But scaling out a pool of Cloud instances for every request received within the system is not a practical solution (Mann, 2015). Such provision can waste the computing resources within the system environment as some resources may remain idle during the operation. A significantly large number of simulations needs to be run to

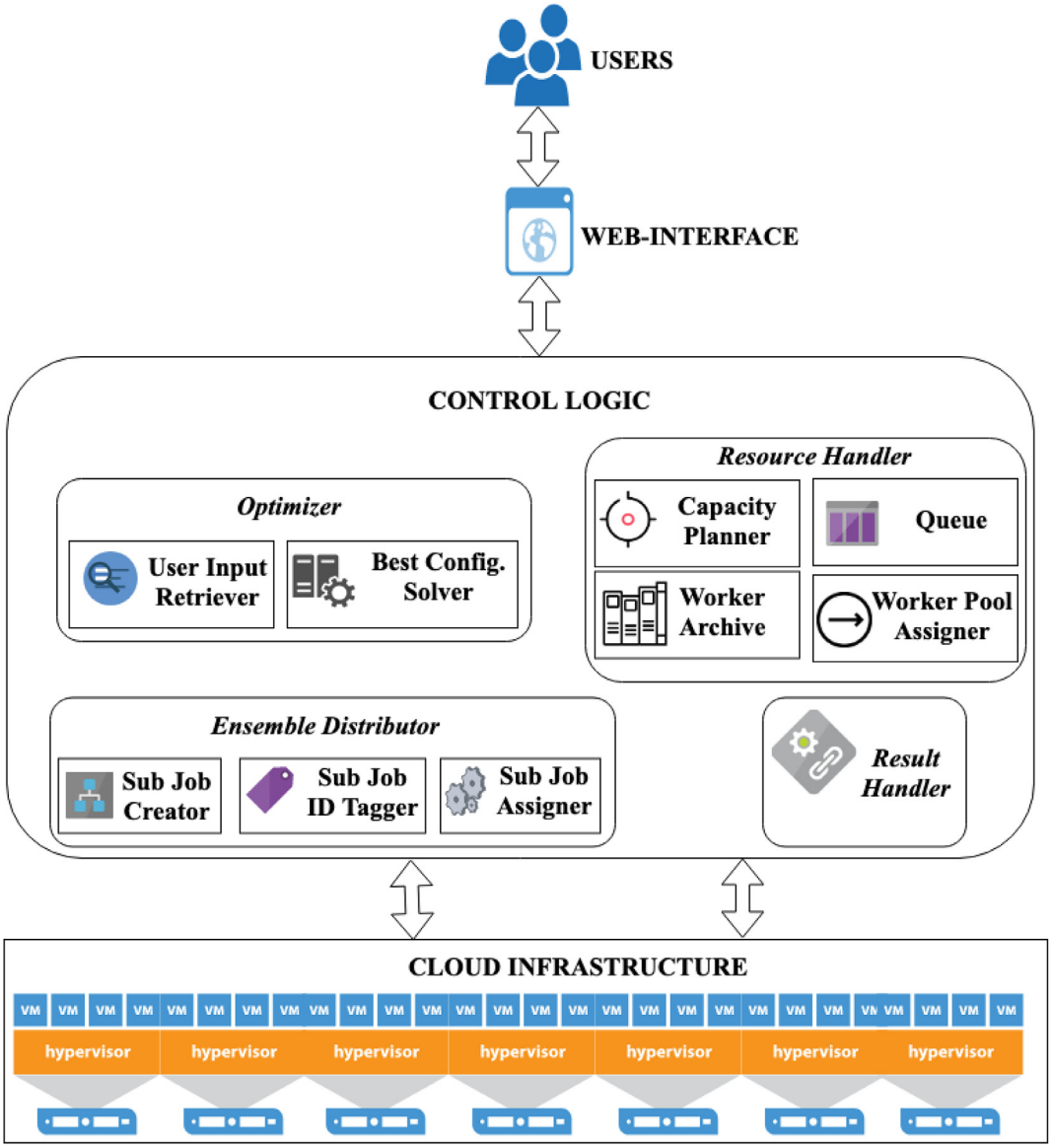


Fig. 3. Component overview of proposed system design.

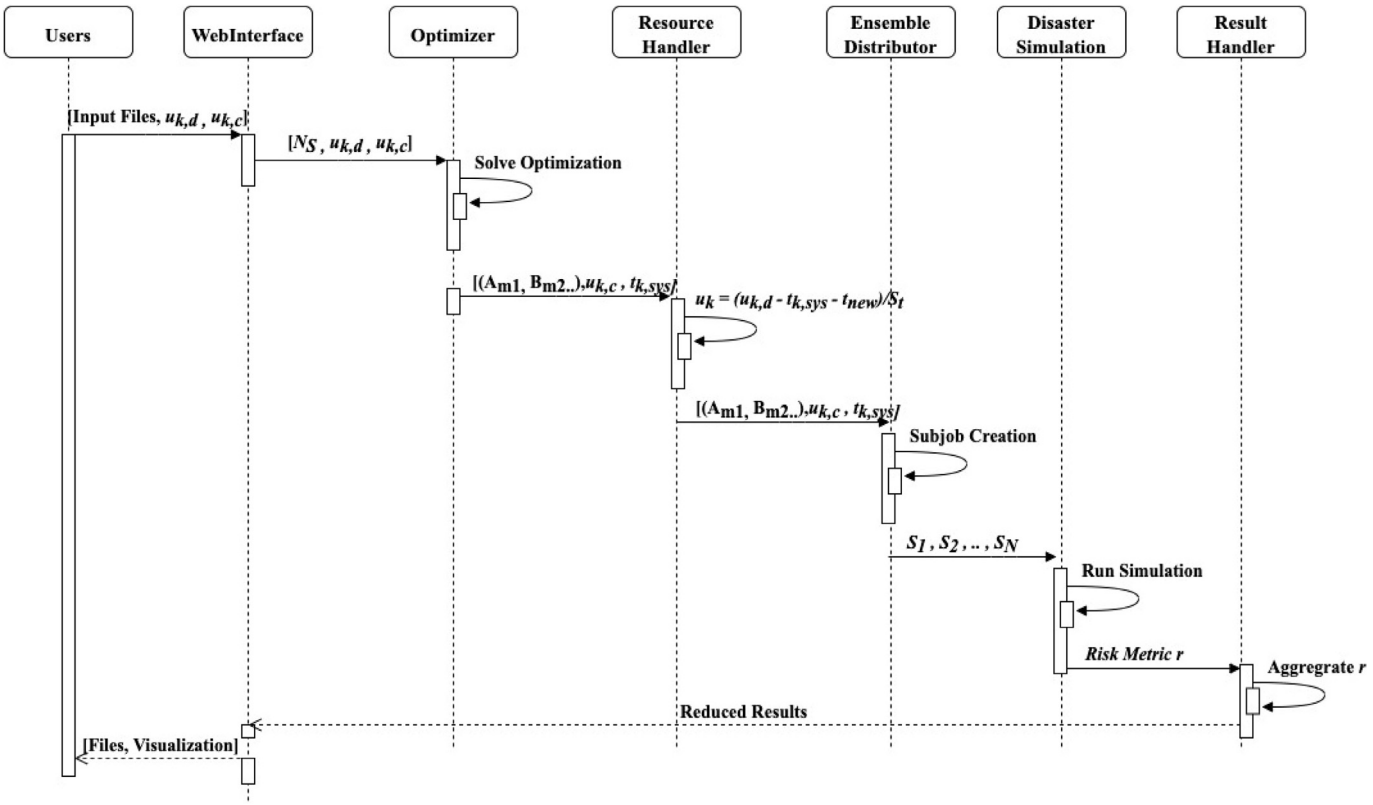


Fig. 4. Sequential overview of the proposed system design (the symbols and notations are listed in Appendix 1).

offer the ensemble of disaster simulations as end services to multiple users. The computative processing for such a large number of simulations can be compute-intensive, and thus, the ensemble has to be broken into simpler groups of simulations, subjobs. Such fractions can independently run in multiple workers in batch mode. It can be a non-trivial task to define a mechanism that provides rational support to execute the computations required by the ensemble. Such a system should also consider all the related constraints and system scenarios at the given instant of the time. The decision to allocate new resources and delete the existing resources from the available pool can be critical. It becomes more challenging when the system has to consider simultaneous user requests from multiple users. Advanced scheduling and optimization mechanisms may be required to ensure the maximum resource utilization while supporting the computational complexity of the ensemble of simulations.

3.2.3. *Trade off between user requirements and cost*

The user requirements have to be considered while offering the ensembles as services to end-users. If required, the user requirements may have to be prioritized, and operations might have to be customized to meet the strict user requirements in terms of time and cost. Moreover, Cloud resources may be massively used as there may be a large number of concurrent users accessing the service. It can be a challenging task to ensure minimal operating cost while complying strictly with the user needs and requirements. The situations dealing with the trade-off between the operational cost and user requirements can be tricky to handle within the system. The diverse range of cost brought in by different pricing models can add more complexity to the trade-off between the requirements and the operating cost.

4. Proposed framework

In this section, we describe our proposed system design (as shown in Fig. 3) that offers the ensemble as end-services by addressing the associated challenges. The system design consists of Users, Control Logic, and Cloud Infrastructure as major entities. Optimizer in the Control Logic takes the user input and requirements entered into the system through a web-interface into consideration to determine the best distribution of simulations for executing the ensemble. Resource Manager accepts the service request with corresponding worker configuration determined by Optimizer. It then selects the cost-efficient Cloud instances strictly based on their urgency level scores, calculated when the requests enter the block. Ensemble Distributor creates several variable-sized fractions of ensembles as subjobs in an orderly fashion before assigning them to the workers in the Cloud infrastructure. Multiple workers execute different runs of simulations to contribute to the ensemble simulations ultimately. The filtered results are collected by Result Collector, which can be accessed by the user through the same web-interface after all the workers have completed their subjobs. The overall sequence of the operations in the proposed system with the message exchange between the components is given in Fig. 4. The system design is explained in detail with its components below:

4.1. *Users*

The users submit a service request along with input files and time and cost requirements through web-interface to initiate an ensemble simulation of the disaster model. The interface contains input fields for the time and cost requirements while the configurations of disaster simulations are defined in the input XML file. A sample of input XML file is shown in Fig. 2. The XML file defines the location where the fire starts, the number of different fire start locations, simulation time and other information related to the input and output data sets. The input files contain the meteorological data and fuel information required for the fire simulation. The configuration defines the location, the number of simulations in the ensemble and input data to be considered for calculation of the risk metrics from the simulation. Web-interface hides all the other steps that are carried out within the framework so as to serve a user

request. The users get to download the result files through the same interface once the execution of the ensemble is completed.

4.2. Control Logic

Control Logic retrieves the user input and requirements and performs several operations through its components so that the ensemble of simulations are optimally distributed among multiple Cloud instances. The components of this entity are further discussed below with their functions.

4.2.1. Optimizer

It employs a user-based policy to manage the multiple user requests in an efficient manner that ensures the user requirements are met with maximum resource utilization. This block uses the retrieved user requirements in conjunction with benchmark records to give the best configuration for the job execution with minimal cost. The series of operations in this block is algorithmically explained in Algorithm 1. Efficient resource utilization and cost is achieved through several subcomponents, which are described below:

User Input Retriever. This component retrieves the user inputs and requirements from the service request initiated by the end-users. It also defines the job complexity in terms of the number of simulations required for the ensemble. The configuration for the ensemble is also retrieved. These requirements are useful for determining the efficient resource for the service request.

Best Configuration Solver. It deals with the efficient creation of variable fractions of the ensemble that ensures the user requirements are met with minimal cost. This component assumes the first of the two optimization tasks in the proposed system design and efficiently creates multiple fractions of the ensemble simulations as subjobs.

While deploying an ensemble of simulations over the Clouds, the ensemble has to be divided into several variable-sized fractions so that multiple workers can independently execute the simulations. The number and size of the fractions are the two most important factors in the deployment, which should be determined based on several constraints. The user requirements have to be considered as well during the deployment of the ensemble as end-services. The availability of different flavors of Cloud instances as workers with varying capabilities of computation is also a constraint in the problem formulation. As such, distribution of simulations in an ensemble to create several variable-sized fractions of the requests can be formulated as an optimization problem that minimizes the incurred cost of operation, as explained below.

Let, M_i be the worker of different flavors/types i, p_{M_i} be the number of worker of type M_i in the best configuration, C_{M_i} be the operating cost associated with the worker type M_i, t_{j,M_i} be the time of operation for worker j of flavor M_i, N_S be the total number of the simulations in the user request, n_{s,j,M_i} be the number of simulation run by worker j of type M_i, T_u be the user requirement of time, C_u be the user requirement of cost, N be the total number of different flavors of the workers.

The efficient distribution of an ensemble for a particular service request k can be formulated as:

$$\begin{aligned} min\ C &= \sum_{i=1}\sum_{j=1} C_{M_i} \times t_{j,M_i} \\ s.t. &\sum_{i=1}\sum_{j=1} n_{s,j,M_i} = N_S \\ &\sum_{i=1}\sum_{j=1} C_{M_i} \times t_{j,M_i} \le C_u \\ \forall j \in \{1,2,\ldots,N_{M_i}\}, i \in \{1,2,\ldots,N\},\ & 0 \le t_{j,M_i} \le T_u \\ p_{M_1}, C_{M_i} \ge 0 & \end{aligned} \tag{1}$$

where, t_{j,M_i} is the time for which the j^{th} instance of flavor type M_i runs and n_{s,j,M_i} is the number of simulations in the fraction which the j^{th} instance of flavor type M_i executes. The first constraint represents the number of simulations required in an ensemble while the second constraint is the related to the user-defined cost such that the feasible operating cost should always be less than or equal to the user-defined cost. The third constraint represents the user-defined time constraint, while the last constraint defines the non-negativity of number and operating cost of the Cloud instances.

The problem has to consider finding an efficient way of assigning the different numbers of simulations to each worker based on its type. This is a complex NP-Hard optimization which cannot be solved within polynomial time. For this study, a heuristic is considered that determines the variables n_{s,j,M_i} from the benchmark experiments using the function $G(T_u, M_i)$ defined as:

$G(T_u, M_i) = \{n : n = Max\{M_i, n\}$ and $n_{M_i} \le T_u\}$

The variable t_{j,M_i} is assigned a constant urgent value deduced after experimental studies. The NP-hard problem now becomes linear and can be solved using existing linear optimization techniques. The solution gives the efficient distribution of the ensemble concerning the best configuration of Cloud instances.

For any user service request u_k with associated requirements of cost $u_{k,c}$ and time $u_{k,d}$, this block gives out the efficient ensemble distribution in the form $[(A_{M_1}, B_{M_2}, \ldots), u_{k,d}, t_{k,sys}]$ where A, B, … are the numbers of Cloud instances of flavor types M_1, M_2,… respectively required in the cluster to execute the request and $t_{k,sys}$ is the time for which the user request u_k has been in the system. This information is passed on to Resource Handler for the allocation of the resources. The working of Optimizer is algorithmically summarized in Algorithm 1.

Algorithm 1. Algorithm for the operation of Optimizer

Input: u_k, $u_{k,d}$, $u_{k,c}$
Output: $[(A_{M_1}, B_{M_2}, \ldots), u_{k,d}, t_{k,sys}]$
1: For every u_k
Retrieve $u_{k,d}$, $u_{k,c}$, N_S
2: Formulate as an optimization problem min C
3: Determine n_{s,j,M_i} using $G(T_u, M_i)$
4: Solve the optimization problem using Linear Optimization techniques
5: **return** $[(A_{M_1}, B_{M_2}, \ldots), u_{k,d}, t_{k,sys}]$

4.2.2. Resource Handler

Resource Handler is the block in the proposed system design that undertakes the second phase of optimization by choosing the most cost-efficient instances based on different Cloud pricing models. The choice of Cloud instances based on pricing models can significantly minimize the cost of operation. The deployment of the ensemble runs on spot instances can incur comparatively lower cost when compared with on-demand instances, but the reliability of such spot instances is less. As such, we introduce three different categories for the user requests-high, medium and low, strictly based on their deadlines (similar to the concept explained in Huang et al. (2013)). A predefined standard S_t obtained from benchmark studies is taken as a reference, and all the user requirements of the deadline ($u_{k,d}$) are compared against the standard to give a parameter, urgency level UL_k given as follows.

$$UL_k = \frac{(u_{k,d} - t_{k,sys})}{S_t} \tag{2}$$

where, $t_{k,sys}$ is the time elapsed after the user request u_k is received within the system.

The urgent requests ($1 \le UL_k < 2$) is directed towards the Capacity Planner, while for other user requests ($UL_k \ge 2$), the creation of new Cloud instances is considered by adding t_{new}, the average time required to create the new Cloud instance, in Eq. (2) and the urgency level UL_k is updated accordingly as follows.

$$UL_k = \frac{(u_{k,d} - t_{k,sys} - t_{new})}{S_t} \tag{3}$$

Table 1
Different urgency levels of user requests.

Level	Values of UL_k
High	$1 \leq UL_k < 2$
Medium	$2 \leq UL_k < 3$
Low	$UL_k \geq 3$

The updated parameter UL_k determines the position of the user request u_k in the queue and which types of instances are allocated to the request. The three defined categories for the values of UL_k are listed in Table 1.

Any service request with a value of UL_k less than *one* is rejected as the request is not feasible. The requests under high urgency level can only be run once in the system and hence are serviced using highly reliable on-demand instances, handled by Capacity Planner. The requests under medium and low categories are served with spot instances with relatively low reliability. If unsuccessful, the requests are rerun with altered urgency level values with more reliable instances. The proposed system does not consider fault tolerance and checkpointing for recovery in spot instances. The working of Resource Handler is algorithmically discussed in Algorithm 2. The components of Resource Handler are discussed further below.

Algorithm 2. Algorithm for Operation of Resource Handler

Input: $[(A_{M_1}, B_{M_2}, ...), u_{k,d}, t_{k,sys}]$
Output: $[u_k, (A_{M_1}, B_{M_2}, ...), D_{type}, bid_{price}]$
1: For every u_k, Calculate $UL_k =$

$$\frac{(u_{k,d} - t_{k,sys})}{S_t}$$

2: **if** UL_k <1 **then**
3: reject u_k
4: **else if** $1 \leq UL_k$ <2 **then**
5: $D_{type} = on - demand$
6: bid_{price} =0
7: Send u_k to Capacity Planner Queue CP_q
8: Push u_k to **R**
9: **else**
10: Update UL_k as follow: $UL_k =$

$$\frac{(u_{k,d} - t_{k,sys} - t_{new})}{S_t}$$

11: **if** UL_k <2 **then**
12: go to Step 4
13: **else**
14: Push the request u_k into the queue **Q** and sort **Q** based on the values of UL_k
15: **end if**
16: **end if**
17: **while** u_k on the top of the queue **Q** **do**
18: **if** $2 \leq UL_k$ <3 **then**
19: $D_{type} = spot$
20: $bid_{price} = bid_{medium}$
21: **else**
22: $D_{type} = spot$
23: $bid_{price} = bid_{low}$
24: **end if**
25: Retrieve the number of free and available workers $n_{M,i}$
26: **for** every M_i **do**
27: Calculate $\Delta n_{M,i} = n_{M,i} - X_i, X = A, B, ..$
28: **if** $\Delta n_{M,i} < 0$ **then**
29: Create $\Delta n_{M,i}$ new Cloud instances of flavor type M_i
30: Update information in Worker Archive
31: **end if**
32: **end for**
33: Forward u_k to Ensemble Distributor
34: Remove u_k from **Q** & Push u_k to **R**
35: **if** u_k == **completed** **then**
36: Remove u_k from **R**
37: Update workers' status in Worker Archive
38: return
39: **else if** u_k == **failed then**
40: go to step 1
41: **else**
42: wait
43: **end if**
44: **end while**

Capacity Planner. This block is included in the proposed system to save time for creating new instances for the user requests with high urgency levels. It keeps track of the rate of the urgent user service requests that are received at Resource Handler in a queue CP_q. In the proposed system, specially for the user requests with urgent deadlines, there must be workers readily available as the time required for the creation of new workers can significantly compromise the urgency of the requests. To overcome this issue, Capacity Planner makes sure that there is at least a minimum number of different workers always available in the system. Capacity Planner can increase the number of already available worker based on the emergency situation and the demand of user requests with urgent deadlines. The additional cost of keeping the cloud instances alive even without any operation can be distributed over the users who initiate such requests. Capacity Planner can use M/M/c (Tijms et al., 1981) queuing model to estimate the number of on-demand instances to be created in advance. For the model, λ is the arrival rate of urgent user requests, μ is the service rate, and c is the number of clusters. For the arrival rate of requests and service rate of the system assumed to follow Poisson distribution, the minimum number of workers of each flavor type M_i required can be determined using Erlang B formula (Messerli, 1972) (Eq. (4)) with very small (nearly zero) value of blocking probability.

$$N_{M_i} = min\{y : B(X, y) \leq T, y \epsilon \mathbb{N} \tag{4}$$

where, $X = \frac{\lambda}{\mu}$ is the traffic offered in Erlang, T is the desired blocking probability (very small) for Capacity Planner, and $B(X, y)$ is the blocking probability expressed as follows.

$$B(X, y) = \frac{\frac{X^y}{m!}}{\sum_{i=0}^{y} \frac{X^i}{i!}} \tag{5}$$

For any instant of time, the number of Cloud instances to be created in advance can be calculated using the historical data (after determining the values of μ and λ). Capacity Planner determines the number of minimum workers required for an almost zero blocking probability in a fixed interval of time (average time for the creation of the new instances). The operation of Capacity Planner for urgent user requests is algorithmically presented in Algorithm 3.

Algorithm 3. Algorithm for Capacity Planner of urgent user requests

Input: $[\lambda, \mu]$
Output: $[N_c]$
1: Forward all user requests u_k in CP_q to Ensemble Distributor
2: For every time interval t (average time required for instance creation), Retrieve updated λ_t, and μ_t
3: Retrieve the number of free and available on-demand workers $nd_{M,i}$
4: **for** every M_i **do**
5: Calculate $Nd_{M,i}$,
$Nd_{M,i} = min\{y : B(X, y) \leq T, y \epsilon \mathbb{N}$
6: Calculate $\Delta nd_{M,i} = nd_{M,i} - Nd_{M,i}$
7: **if** $\Delta nd_{M,i}$<0 **then**
8: Create $|\Delta nd_{M,i}|$ new on-demand instances of flavor type M_i
9: **else**
10: Delete $\Delta nd_{M,i}$ on-demand instances
11: **end if**
12: Update information in Worker Archive
13: **end for**
14: Wait until the end of time interval t
15: Go to Step 1

In addition to the urgent user requests, this component is useful in deciding when to reserve the Cloud instances to further minimize the cost based on the historical records. For example, for wildfire ensembles, based on the historical information about the arrival rate of user requests, Capacity Planner can reserve a pool of instances during the summer.

Queue. It keeps the record of all the user requests with the corresponding efficient ensemble distribution scheme given by Optimizer. For each user request u_k, urgency level UL_k is calculated. The queue stores all the user requests in a sorted manner such that UL_k with lower values are placed on the top. The required Cloud resources are allocated to the requests on a one-at-a-time basis. There is an additional queue R which keeps the record of all the user requests being serviced with corresponding $t_{k,sys}$.

Worker Archive. It keeps a record of all the workers within the proposed system. The information about the flavor, pricing model and availability of the worker is essential for effective resource allocation. For any cluster size requested by the service request, this component provides the information about the availability of the workers running in the system to prevent the creation of new instances if not required.

Worker Pool Assigner. It decides to deploy the cluster of Cloud instances based on different pricing models strictly based on the category defined by the values of UL_k. This component handles the trade-off between the urgency level and cost by altering the reliability of the instances accordingly. If the user request has an urgent deadline, Worker Pool Assigner opts on-demand instances with higher reliability. Worker Pool Assigner bids for spot Cloud instances for the job in medium and low categories with bid prices $bid_{medium} > bid_{low}$ established based on historical information. If the requests in medium and low categories are not completed due to the unavailability of the spot instances, the requests are pushed into the queue Q with an altered value of $t_{k,sys}$.

4.2.3. Ensemble Distributor

Worker Distributor handles the creation and distribution of the variable-sized fractions of the ensemble initiated by the user service request following the cluster size and type defined by Optimizer and Resource Handler. For a worker W_{M_i} of flavor type M_i, Ensemble Distributor retrieves the number of simulations in a process n_{sp} and the number of simultaneous processes of the disaster model x_p and assigns the corresponding fractions to the workers. Depending on the computational capability of the instances, the worker nodes may or may not implement multiple processes of prediction software tool simultaneously. The functionalities of this block are algorithmically represented in Algorithm 4. Worker Distributor in turns consists of the following components:

Subjob Creator. It creates several subjobs with variable sizes based on the configuration given by Optimizer. All the subjobs possess the characteristics of the main job and can be run in an independent mode. The last subjob created by the Subjob Creator compensates for any additional number of simulations in the best configuration by assigning a lesser number of simulations to the worker under that particular subjob.

Subjob ID Tagger. It adds identification tags to all the created subjobs before assigning them to the workers. The information tags are received and decoded for customizing the simulation runs for contributing to the specified fraction of the entire ensemble run.

Subjob Assigner. After addition of the identification tags, Subjob Assigner assigns respective subjobs to the corresponding workers in the cluster. The last subjob that compensates the over-estimation of the best configuration is chosen such that the operation cost is reduced for the service request. All the necessary files required for the execution of the prediction software tool are downloaded in the worker nodes from the master controller within the system environment.

Algorithm 4. Algorithm for work division and distribution

Input: Cluster Size, N_c
Output: Intermittent Result Files
1: Retrieve n_{sp} and x_p for each W_{M_i}
2: Create S_N subjobs where subjob S_N acts as compensating subjob with possibly less number of simulations
3: Add subjob identification tags # to subjobs
4: Assign x_p subjobs as different fractions to corresponding worker W_{M_i}
5: Assign compensating subjob S_N to the least costly worker W_N in accordance to the configuration given by Optimizer
6: Wait until all subjobs are completed
7: Reduce the result files
8: **return** *Reduced Result Files*

4.2.4. Result Handler

During the execution of simulations in the workers, multiple output files are created at the end of each simulation run in different formats after processing a more significant amount of relevant data. The transfer of the entire simulation results back and forth between the worker nodes and the master node can create a network bottleneck, thereby compromising the performance of the system. As such, Result Handler makes sure only the significantly important information is extracted out from the outputs generated after every run of the simulation. The reduced but important output information is gathered in a centralized fashion under a single folder that references to the subjob identification tag. Upon completion of the execution of the subjobs, only the critical information set with relatively small data size is sent back to the master node. The master node stores the files in a centralized fashion. After successful uploading of the data to the master instance, the worker nodes delete all the files related to the completed job and make themselves available to take new subjobs. When the master node receives all the relevant output files from the worker nodes under a single folder referencing to the main job, the job is deemed to be complete. Upon completion of the main job, the users can see the status reflected in the web interface and download all the output files for further interpretation and visualization.

4.3. Cloud Infrastructure

Cloud Infrastructure uses public Clouds to provide required hardware foundation in terms of virtual machines of different flavor types to support the computational needs of ensemble simulations. All the workers have Spark tool pre-installed on them that run different processes with different start points to contribute to the ensemble simulation as initiated by the service request.

5. Evaluation

The working of the proposed system design is validated through a real prototype which utilizes Spark, a wildfire simulation tool that predicts the progression of a wildfire. Spark offers a modular framework for wildfire spread prediction where several packages and models can easily be plugged in. These packages and models include generation of wind fields and their topographic correction, ignition models, fire-line interactions, road and transmission models and firebrand transport (Miller et al., 2015). All the calculations required for a fire simulation in Spark are parallelized on Graphical Processing Unit (GPU) architecture such that the simulations can run faster than in real-time. This is true for all the simulations that aggregate in an ensemble to give more accurate risk metrics of a fire.

All the steps explained in the proposed foundation system are closely

followed during the evaluation. The proposed solution provides modular system design that offers flexibility to change the components (e.g. wildfire simulator with other disaster simulation tool). Java is the main programming language used to enable different mechanisms within the system. A web-based user interface is developed to facilitate the users to access the system and initiate the request to use ensemble simulations as end services. The web-interface to upload the files and enter the user requirements of time and cost is shown in Fig. 5.

In the following section, we first give the details of the use case scenario and Cloud infrastructure that are utilized for validating the proposed system. Then, the results for benchmarking of Spark in the Cloud environment are presented and discussed. The critical parameters required for subsequent operations in different blocks are determined through the benchmark studies. Based on these results, *Optimizer* decides how to create multiple fractions with variable size to contribute to the ensemble required by the users. The influence of user-defined deadlines on the choice of instances based on different pricing models and subsequently on the total cost of operation is also studied. Finally, we evaluate the overall performance of the proposed system against the comparable on-premise system and bag-of-task type execution over the Clouds.

5.1. Ensemble use case scenario

For evaluation, a real ensemble scenario using data kindly provided by the Tasmania Fire Service (TFS) is used. The scenario consists of a total of 169 simulations starting at equally spaced locations 1 km apart on a 13 km × 13 km grid around a central point. Each simulation is configured to run for 9 h after the fire has started at a point. The model is configured for various fuel types in Tasmania and also takes into account impact with any urban areas by counting the number of urban cells burnt for a particular wildfire. This prediction model falls into a risk modeling category of ensemble simulations analyzing the risks of a wildfire starting at an unknown location under a particular set of weather conditions. This fundamental design can be further extended to work for operational modeling that deals with direct suppression and evacuation efforts once the fire has been reported to start.

5.2. Setting up the cloud environment

In this experiment, Nectar Cloud (nec), an OpenStack-based community Cloud infrastructure, is used as an emulated Amazon Cloud environment for conducting different experiments. It is clear from the benchmark studies that the number of cores in the Cloud instances is the key factor in determining the time taken to run a fixed number of simulations. All the available cheapest instances with their hardware specifications along with their unit cost are listed below in Table 2. The cost of operating the instances is set according to the Amazon Web Services (AWS) (ama) pricing model. The data transferred into Amazon Cloud and data transfer between the instances in the same availability zones are free. Thus, data transfer cost is not taken into consideration in this study. But, the time taken for the data transfer is considered for total operation time, and hence, the time taken for data transfer contributes to the total operating cost and time. As for the spot instances, resources are abruptly taken out from the system design during operation with the probabilities calculated using existing works.

Table 2
Different flavors in nectar cloud.

VCPUs	Flavor	RAM (GB)	On-demand cost ($/hr)	Spot cost ($/hr)	Reserved cost ($/hr)
1	m2.xsmall	2	0.0146	0.0035	0.01
2	t3.small	2	0.0209	0.0051	0.0142
4	t3.medium	4	0.0418	0.01	0.0284

5.3. Benchmarking of spark over cloud environment

For the natural disaster simulations like fire simulations, the time taken for each simulation is dependent on several factors and has not been previously studied. Creating several batches without a general understanding of the fire dynamics can contribute to inefficient operation in the proposed system design. Given the parallelization of the simulations in Spark, independently accommodating simultaneous Spark process can enhance the resource utilization. As such, we conducted a set of different experiments under the benchmark study to determine the efficient distribution of the simulations in the ensemble based on the processing capacities of the workers. For all the different flavors of instances available in the Cloud environment, we analyze the implications of the processing capabilities and cores in the total execution time. The benchmark tests were carried out in two distinct phases - first with the different number of simulations in each instance and later with several simultaneous processes of disaster model in the instance. Moreover, the key parameters S_t and n_{s,j,M_i} are also determined after the experimental analyses.

5.3.1. Number of simulations

For each of the different instance flavors, a series of experiments was carried with different fire start points with a batch of variable size of the simulations in each worker. For the TFS sample, there are 169 different geographical start points for the fire. The fires are started on a regularly spaced grid at 1 km intervals irrespective of the land classification. It should be noted that fire simulations starting in areas of water take

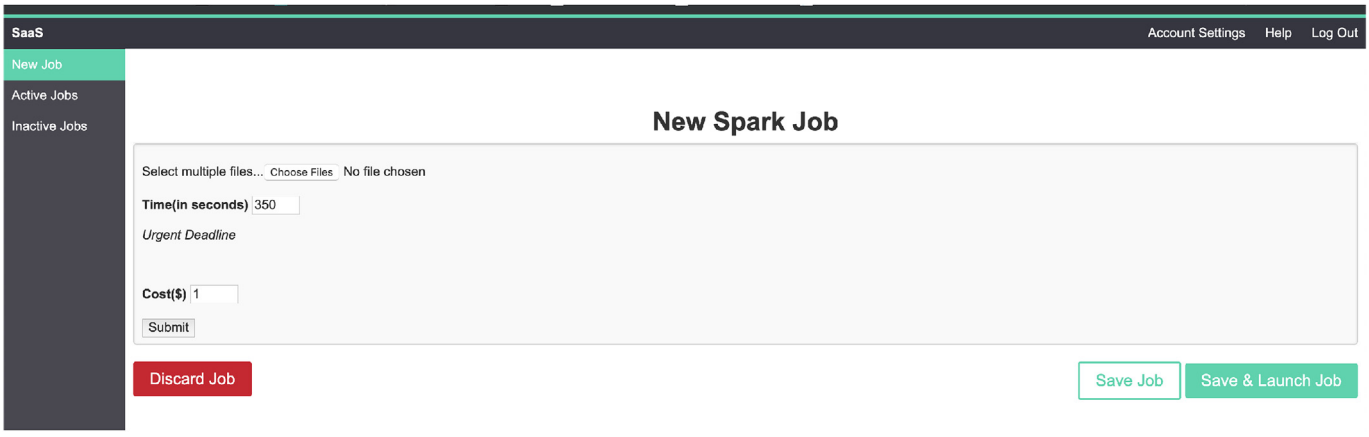


Fig. 5. Web-Interface to initiate request in proposed system.

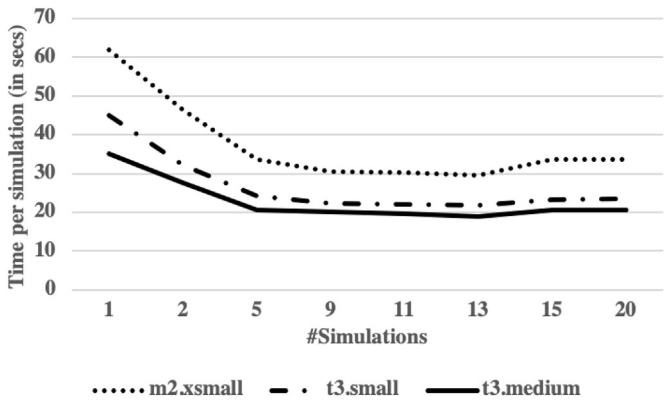


Fig. 6. Unit simulation execution time for different worker flavors.

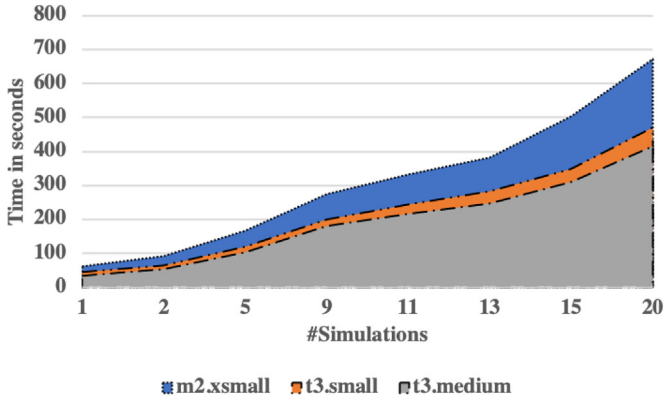


Fig. 7. Total simulation execution time for different worker flavors.

significantly less time (as they terminate immediately) compared to the simulation on land. For every sample file, the experiments are carried out in six distinct sets of 1, 2, 5, 9, 11, 13, 15 and 20 simulations in each unit of workers. The findings for all the instance flavors are figuratively presented in Fig. 6. Moreover, the average times taken by the worker instances to complete the different sets of simulations with a single process of Spark running are depicted in Fig. 7.

Due to the more significant computation resources in t3.small and t3.medium workers, as compared to the m2.xsmall worker, the average unit execution time for a Spark simulation is lower in t3.small and t3.medium. Knowledge of this difference is useful while choosing the cluster of workers for a user service request with different sets of inputs. The execution time per simulation decreases when executed in batches until the saturation point (different for different instance flavors). The average time per simulation keeps improving until the set of 13 simulations for all instances and saturated after that with a slight increase. This improvement is due to a common data fetch cycle for all the simulations which can be done once when executed in a batch compared to multiple times when executed as independent units. The findings of the benchmark study show that the time performance of the system improves when simulations are executed as a batch (variable) rather than when executed independently in different machines. For a single Spark process, the execution time of the simulation increases with the increased number of simulations in the batch. The time performance of the instances beyond the saturation points is out of the scope of this study. Moreover, the improvement in the time performance of the instances is not linear with the increase in RAM size, as shown in Fig. 7.

5.3.2. *Simultaneous operation of spark processes*

The disaster model, Spark, consists of two cycles - data paging and computative processing during the execution of fire simulations. Since there are large data sets involved in the process, there is a possibility of the process sitting idle while the large data sets are paged from storage into memory. Due to this, we evaluated the feasibility for running different batches in a single worker to ensure maximum resource utilization within the system environment. For all the instance flavors, tests were carried out in a way such that multiple subjobs are assigned to a single worker for simultaneous operation. Under such an operation, the worker has to execute the different Spark processes with different start points contributing to the ensemble. In a trial and error fashion, we related the total number of simulations a worker can support, for a given deadline, with the varied number of VCPUs available in the workers. Moreover, the effects of using more processor memory in the execution are compared against the performance gain achieved by accommodating multiple model processes in an instance with multiple VCPUs. The multiple subjobs run on a single machine in an independent under different configurations and the time performance of the instances were recorded for further analysis.

The time performance of different instances for multiple processes of Spark is depicted in Fig. 8. In the figure, n_1 is the number of simulation in a single process, x is the number of Spark processes, x_n is the number of simulation in each Spark process, and $N_{S,x}$ is the total number of simulation for x Spark processes in a single machine. A single Spark process consists of two main sub-processes that are CPU-dependent computations and disk/network-dependent data operation. The efficiency of the worker nodes can increase significantly if the computation sub-process can be overlapped with the data operation of another simulation. The presence of a single processor is unable to complement the data fetch and computation cycles. It is thus, clear that the instances with a single VCPU are not able to support the multiple processes of Spark. The time performance keeps improving until N processes are accommodated in the instances with N VCPUs, which facilitates the system to accommodate more simulations in a fraction of the ensemble. For a deadline of 300s, the worker of t3.medium type with 4 VCPUs can run four simultaneous processes of Spark with a total of 32 different simulations compared to the run of three simultaneous processes with a total of 30 simulations and single process with a total of 15 simulations. When five simultaneous processes are run on the instance, there is no improvement in the total number of simulations that can be run. Based on the findings, we establish a fact that N simultaneous processes of Spark can be run a Cloud instance with N processors for optimal performance.

The performance gain, due to the increasing the number of VCPUs, out shades the same due to increased RAM sizes in the instances. Moreover, for a constant number of VCPUs in the instances, the increase in the RAM sizes does not significantly increase the total number of simulations. Thus, we focus on the most cost-effective instances with a varied number of VCPUs without any regard to the RAM sizes.

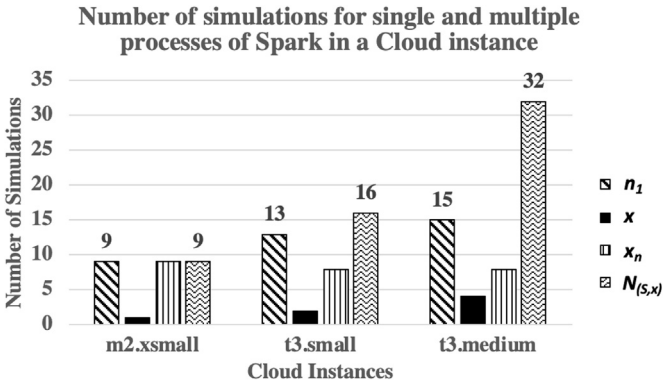


Fig. 8. Number of simulations for multiple processes of Spark running in the instances (Time: 300 s).

5.3.3. Determination of S_t and n_{s,j,M_i}

The set of 169 simulations in the ensemble was run with different locations sequentially over instances with different flavor types. The value S_t was fixed at 300 s by considering the fact that the average time to run the ensemble over the most powerful machine is 3912 s. The value S_t can easily be adjusted to make the system more responsive to the user requests. The corresponding values of n_{s,j,M_i} were then obtained from the experiments conducted in the first two phases of benchmark studies. For example, the value of n_{s,j,M_i} is 32 for $T_u = 300$ s and $M_i =$ t3.medium as retrieved by using the function $G(T_u, M_i)$.

5.3.4. Approximation of minimal time and number of simulations for an instance

Given a set of options for the Cloud instances available, it is always a non-trivial task to accurately estimate the time taken to execute a particular number of simulations and vice-versa. The number of simulations that can be executed by the Cloud instances increases with the increase in the size of RAM when a single process of disaster model is run. The size of RAM does not have a significant impact when multiple processes of the models are run in the instances. As such, we use linear regression to define a relationship between the number of cores, time and number of simulations to provide an approximation of time-based on the configuration of the instance. It should be noted that the accuracy of the approximation is not the primary focus of this study, but the cost optimization based on the results obtained from the approximation is. As such, different advanced methods can substitute the linear regression module to improve the accuracy of the approximation.

5.4. Experimental setup for evaluation of the proposed system

As previously discussed, the objective of the proposed system is to enable the ensemble of natural disaster scenarios as end services with minimal cost achieved through two phases of optimization. To evaluate the performance of the proposed system, we compare the incurred operating cost and time against the ones incurred in an on-premise system and bag-of-task type executions. For an on-premise system, we consider a single machine with the same hardware configurations as the Cloud instances have. Consequently, we have three different on-premise systems with Spark pre-installed on them. We then consider a conceptual idea of bag-of-tasks (BoT) in a distributed environment where each simulation in the ensemble requested by the user is considered a task and executed in as many machines. To compare the resource and cost optimization achieved by the proposed system, we further consider an adaptation of tasks clustering mechanism, as explained in Muthuvelu et al. (2013) for a distributed environment. In what we call the adaptation as *modified bag-of-tasks (mBoT)* execution, equal-sized clusters are formed based on the grid size of the configuration and the job complexities. For example, for a grid size of 13 km × 13 km which yields 169 simulations, 13 clusters with 13 simulations are created. The cost and time performance of the proposed system are compared accordingly against that of the *mBoT* execution.

Table 3
Complexity of user requests.

Label	Grid size (km × km)	#Simulations	Batch size (mBoT)
small	5 × 5	25	5
medium	9 × 9	81	9
large	11 × 11	121	11
TFS	13 × 13	169	13
2 × TFS	26 × 26	676	26
3 × TFS	39 × 39	1521	39

Note: The batch size is 1 for BoT execution, while the batch size is variable for the proposed system.

5.4.1. Evaluation metrics

Operation Cost. The total operation cost in the proposed system design is the cost incurred to run the ensemble simulation over the Cloud environment. The cost is referred to as *Ensemble Service Cost*, which is the cost calculated taking the actual duration for which the workers are in operation while serving the user service request. The cost is calculated on a "per second" basis based on the AWS pricing model as listed in Table 2 using a basic unitary method.

Operation Time. The operation time for a user request is the total time elapsed after the user submits the request to the system until the user gets the result files back. The operation time takes the time taken to upload the required files for the ensemble to the Cloud environment into consideration and is reflected accordingly in the total operation cost. The operating times for multiple workers allocated for a single user request can be different. The operation time for the user request is the maximum of the operating times for each worker allocated for that request.

5.4.2. Experimental scenario

Different levels of user-defined deadlines. For the user requirements of time, we consider three different levels of the deadline, namely High, Medium and Low are considered, as shown in Table 1. The experiments are repeated for five random values in each range to study the influence of urgency level on the total operating cost, and average values are presented.

Complexity of the user request. The TFS samples for wildfire propagation simulation consists of a grid of 13 km × 13 km spaced at 1 km, comprising of 169 simulations for the ensemble. To validate the effectiveness of the proposed system design, we conduct various experiments considering other sizes of the grid.

(5×5[*small*], 9×9[*medium*], and 11×11[*large*]) in the sample files for the comparison against the on-premise system. For comparison against the bag-of-tasks type execution, the sizes (2× TFS configuration [26 × 26] and 3× TFS configuration [39 × 39]) are considered, which are listed in Table 3.

6. Results and discussions

In this section, we discuss the results obtained while validating the proposed framework under different experimental scenarios of user requirements of time and complexities. We also present the comparative analysis of the performance of the proposed system with an on-premise system and the existing state of the art concepts of bag-of-tasks executions and job clustering. Besides, we also present a brief performance analysis of the proposed system under multiple simultaneous users with urgent deadlines.

6.1. Proposed system vs. on-premise setup

For an on-premise setup, the ensemble was run on the instances of each flavor type in a sequential manner. The same sets of the ensembles were run on the proposed system with a high level of urgency. Fig. 9 represents the comparison of the cost incurred when the ensemble is executed using the proposed system and on-premise setup. The on-premise system is painstakingly time-consuming as the efficiency achieved by running the simulations in batches ceases after the saturation point. The comparison of operation time is shown in Fig. 10. On the other hand, the proposed system distributes the ensemble to multiple workers that operate within the optimal performance configurations and produces the desired output in a time-efficient manner. The cost incurred by the on-premise system with the cheapest machine (with the configuration of the cheapest Cloud instance), is the minimum of all. The proposed system operates the workers in their optimal performance region and hence achieves the operating cost closer to the on-premise cost. As such, the proposed system can offer the required services with the cost comparable to on-premise cost but with much improved time efficiency. There is no further cost minimization when the ensembles are run on the

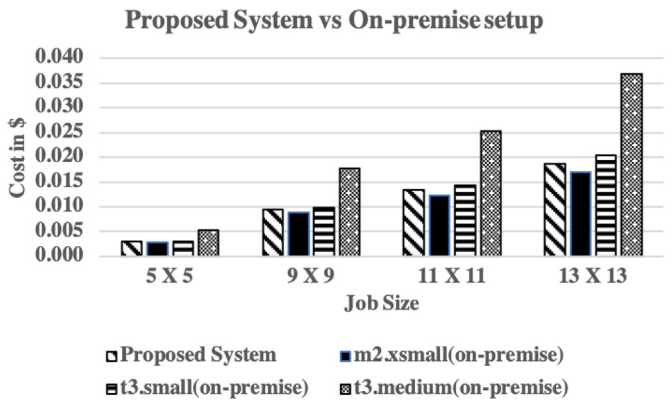


Fig. 9. Cost comparison between proposed and on-premise system.

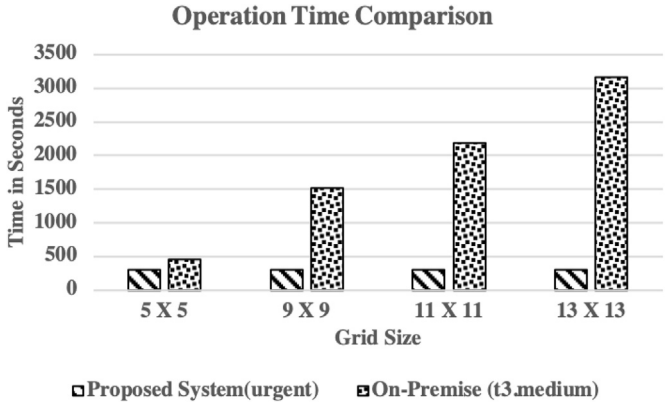
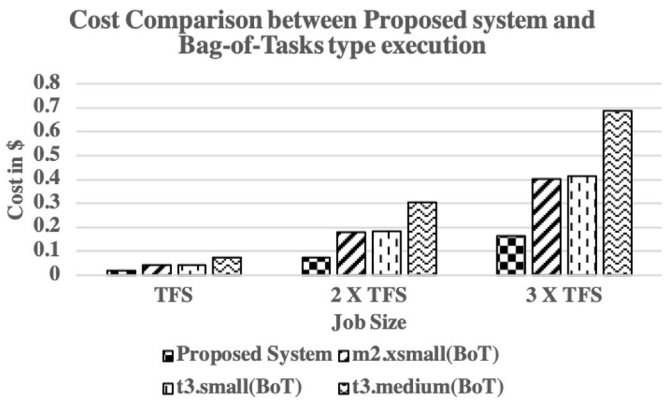
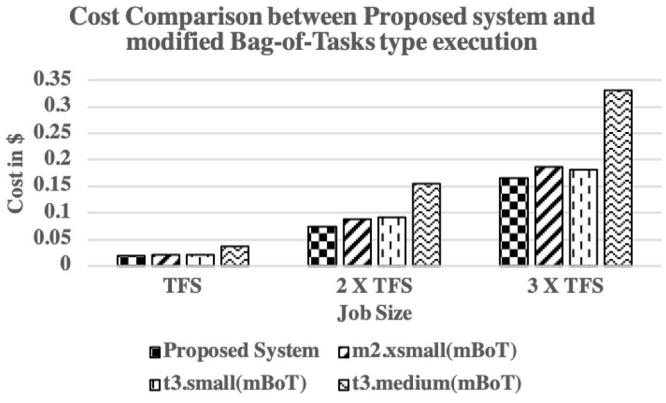


Fig. 10. Comparison of operation time between proposed and on-premise system.



(a)



(b)

Fig. 11. Cost comparison between the proposed system and (a) bag-of-tasks (BoT) execution and (b) modified bag-of-tasks (mBoT) execution.

on-premise machine with better configuration as the resources are under-utilized. The operation cost is up to 98% more than the proposed system, as shown in Fig. 9. The proposed solution ensures the resources are used optimally to avoid such under-utilization. Besides, the end-users get added benefits from the proposed system as the system setup, configuration, and dependencies are well-handled. The same could be a cumbersome task in the on-premise system.

6.2. *Proposed system vs bag-of-task execution*

The simulations in an ensemble are independent units of work without any dependencies among themselves. Thus, for bag-of-tasks type of execution, we consider each simulation as a task and run them independently in a single machine. Fig. 11a shows the comparison between the cost incurred within the proposed system and bag-of-task type execution. The bag-of-tasks execution runs the simulation for a lesser time, but on the other hand, the execution incurs significantly high cost (131%–316%) more than that of the proposed system). The large data sets required for each unit of simulation have to be fetched into the workers. Consequently, the improvement in time performance and resource utilization brought by the running the simulations in batches is non-existent when each simulation is run independently in separate machines.

Moreover, we divide the total number of simulations in an ensemble into several subjobs with an equal number of simulations. Each subjob is considered to be a unit of work and run in as many workers in a modified bag-of-task execution. Instances of all three flavor types are considered for the modified bag-of-task execution. Fig. 11 shows the comparison of the cost incurred in the modified execution and the proposed system. The cost incurred in the modified execution is 9%–108% more than that of the proposed system. The finding reflects the fact that the execution of an equal number of simulations in different fractions in an ensemble is not the optimal way of running the ensemble. The cost-efficiency of the proposed system over the modified bag-of-tasks type execution increases significantly with the increase in the total number of simulations in the ensemble. The execution of the simulations in variable-sized fractions utilizes the versatility of the available workers. Hence, it is possible to further optimize the operating cost by choosing cost-efficient workers in terms of the simulations.

Fig. 12 shows the comparison of time performance between the two systems along with conventional bag-of-tasks execution. The simulations in the ensemble, when considered independent and run over as many workers as the number of simulations, produce the outputs in less time but incurs high cost. The operation time is variable in the modified bag-of-tasks execution, which assumes the equal size of the batch while the operation time in the proposed system is dependent upon the user requirements. When the size of the job increases, the modified bag-of-tasks type execution takes more time as shown in Fig. 12 (For 3 × *TFS* job, the operation time is about 176% more than the proposed system). For the urgent user requests, the system does not have to consider the additional time for the creation of the new instances. In contrast, for similar bag-of-task executions, there is always the time of creating new instances added in the total operation time. Consequently, as shown in Fig. 12, the total operation time of urgent requests in the proposed system is always less than that of the modified bag-of-tasks type executions. Moreover, for other urgency levels of the requests (medium and low), the proposed system solves the trade-off between the time and cost by minimizing the cost to the maximum possible extent.

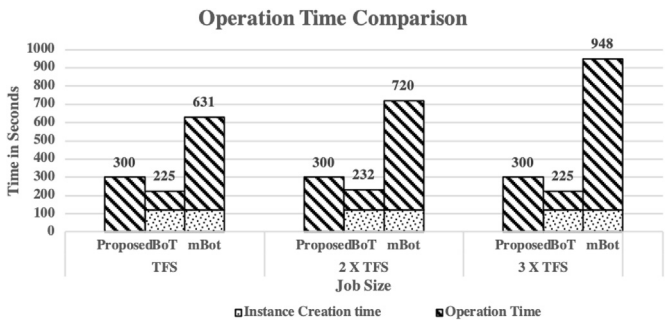


Fig. 12. Operation time for different execution methods.

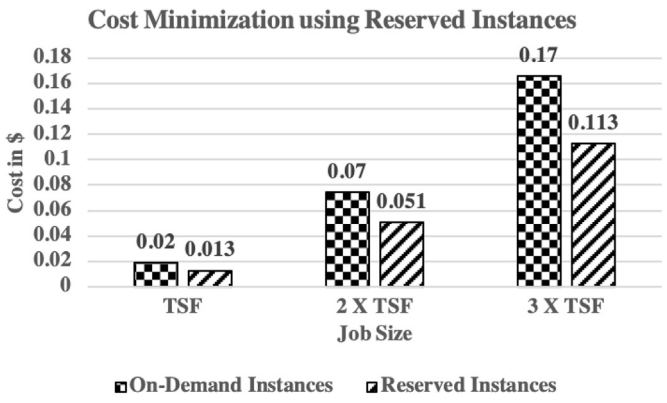


Fig. 14. Cost minimization using reserved instances.

6.3. Cost reduction using Resource Handler

Resource Handler in the proposed system minimizes the operation cost by intelligently choosing the cost-effective instances based on the urgency levels calculated for each user request. The on-demand instances offer higher reliability as these instances are dedicated to the user request once allocated, until the completion of the subjob. The spot instances provide cheaper options for execution, but the reliability offered by these instances is lower. The spot instances are offered to the other users with a higher bid in the Cloud environment, even if the current execution of the subjob is not complete. In this study, the bid prices for the medium and low urgency levels are derived from the historical information issued by different Cloud providers. It is to be noted that the calculation of bid amounts to ensure high reliability is not the aim of this study.

Fig. 13 reflects the possible minimization of the operating cost by deploying the ensembles on spot instances rather than on on-demand instances whenever possible. The user requests with high urgency level were executed on on-demand instances, while those with medium and low urgency levels are executed on spot instances with different bid amounts. For the user requests with low urgency level, the users can minimize the cost up to 73% compared to the requests with the high urgency level. For the requests with the medium urgency level, the cost minimization is up to 76%. This cost minimization is possible due to the trade-off between the reliability and operation cost of the instances. If the proposed system has to abandon the spot instances to other users in the Cloud environment because of higher bids, the system adds those user requests into the queue with altered urgency levels. The recovery and fault-tolerance techniques can ensure the execution of the subjobs getting resumed from the point where they were interrupted, but these techniques are beyond the scope of this work. If medium and low urgent labeled requests fail in the first round, the incurred operation cost is likely to increase. To overcome this cost discrepancy, a cost model that calculates the operating cost based on the request complexity and user deadline can be introduced.

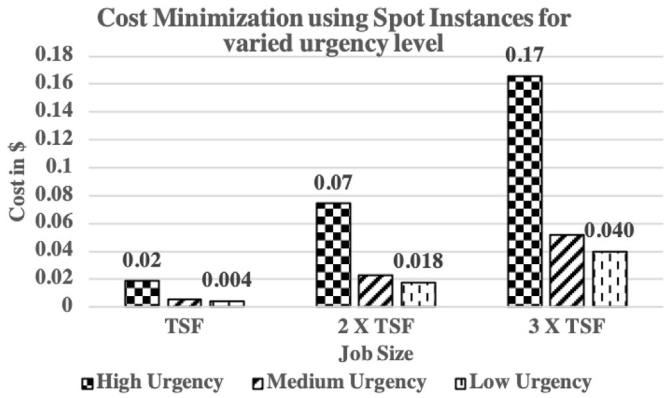


Fig. 13. Cost minimization using spot instances.

Moreover, Capacity Planner in Resource Handler keeps track of the urgent user requests received at the system based on the time. The tracker assigns "peak" label to the duration based on the historical records. The proposed system reserves the Cloud instances in advance for the peak duration, which can further minimize the cost for requests with high urgency levels. The experimental results show that the cost for the user requests with high urgency levels was minimized by about 32% as depicted in Fig. 14.

6.4. Cost vs levels of deadline

The Ensemble Service cost generally increases with an increased level of urgency in the user requests. The total cost of operation is calculated based on the actual time for which the workers were in operation. The cost incurred in the proposed system for different levels of user-defined deadlines is less than the cost incurred in the bag-of-tasks system and close to the cost incurred by an on-premise system with the cheapest machines. The increase in urgency level incurs a higher operation cost (see Fig. 13). The urgent requests have a higher cost and higher reliability as the reliability is traded against the cost. When compared to urgent request, the medium and low urgency incur up to 69% and 73% lesser operation cost. Moreover, the medium urgency incurs about 28% more operation cost compared to low urgency based on different user request complexities. For the requests with medium and low urgent level (based on the values of UL_k), the Ensemble Service cost increases if the service of the spot instances allocated for them is abruptly interrupted by the Cloud provider because of higher bids from other users (not in the proposed system). In the worst case, the user request with a low urgency level can incur the same cost as the request with a high urgency level. This cost discrepancy can be solved by developing a cost model that charges the requests based on the urgency level and the job complexities.

6.5. Cost vs complexity of user requests

The total number of simulations in the user requests can be altered by changing the size of the grid for the wildfire simulation in TFS samples. The operating cost for user request increases with increase in the grid size in the ensemble configuration, which ultimately increases the total number of simulations in the user request. Even for the varied number of total simulations, the proposed system design yielded minimal operating cost which is always less than the cost incurred by the bag-of-tasks execution. The operation cost is close to the cost incurred by the on-premise system with the cheapest machines.

6.6. Analysis of time performance under multiple urgent user requests

To validate the support of multiple simultaneous users, we considered

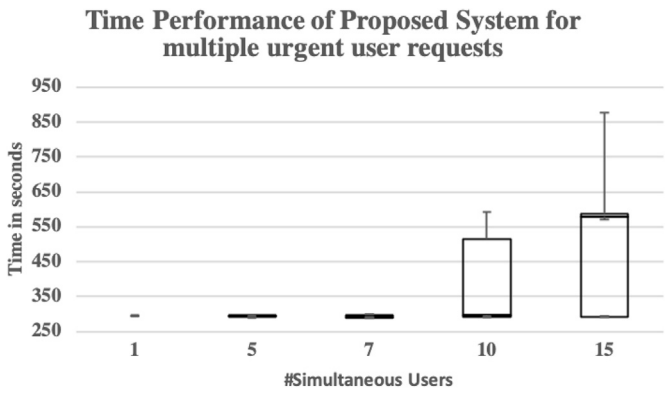


Fig. 15. Time performance analysis under multiple simultaneous users with urgent deadlines (TFS Configuration with 169 simulations).

several simultaneous users submitting the requests (TFS configuration) roughly at the same time. The experiment was conducted with a maximum number of 150 VCPUs in the Cloud environment. Consequently, when more than seven urgent requests are received in the system, the variable batches of three service requests have to wait in the queue. In theory, Cloud infrastructure with a large number of computing nodes would not have any limitation. In the proposed system, when the urgent requests have to wait, the value of UL_k for each request becomes less than one and would otherwise be rejected as infeasible request failing to meet the deadline. For this analysis, we consider waiting time for the requests unable to find free resources. The waiting time contributes to the total time required for serving the requests. For 10 and 15 simultaneous urgent requests, the maximum time taken for serving the requests were 592 and 878 s, respectively (as shown in Fig. 15), including the waiting time in the queue. Nevertheless, the actual time for which the simulations were run is comparable to the time taken to serve at most seven simultaneous service requests. The total cost calculation does not consider the waiting time. Consequently, the user requests with the same complexity with similar deadlines have comparable operation cost. This limitation which requires the requests to wait in the queue, in the proposed system, can be overcome by adding more computing nodes in the Cloud environment during the peak disaster season.

7. Conclusions and future works

Natural disasters like wildfires are a global problem and require accurate and timely simulation for operation prediction and risk assessment. Both of the use cases, presented in the paper, require the ability to efficiently schedule and launch an ensemble of simulations within a resource or time-constrained envelope. Providing the ability to deliver these ensemble simulations as an end service on the Cloud has many challenges, and the delivery and implementation of such a system have not previously been fully explored. This study has proposed and demonstrated an implementation designed for this purpose. The validation results are quite promising with operating cost comparable to conventional and cheapest on-premise setup and up to 300% when compared to bag-of-tasks type execution.

The Cloud-based framework, as presented in the paper, is proposed as a foundation system design with modular blocks which can be improved for more advanced functionalities. In theory, the Cloud infrastructure in the proposed system can support any number of simultaneous users. But, in real practice, it can be limited by the total number of computing nodes available. The simulations in an ensemble are parallelized such that each simulation can be executed independently in its respective batch. The creation of variable batches can save time for data paging for multiple simulations, but the same is not true for the computative processing involved in each simulation. Consequently, the time performance per simulation in a batch saturates at a certain point after which the parallelization does not yield any improvement. The paper discussed trade-offs between cost, urgency level and user request complexities. But, the proposed system cannot deal with cases caused by different circumstances, such as failure of Cloud instances, too many urgent user requests and unavailability of computing nodes). The trade-off between the reliability and the cost can be further studied to make better use of spot instances for user requests with relaxed deadlines.

In the future, the possible cost discrepancy because of the low reliability of the spot instances will be addressed by introducing a cost model based on the user requests (complexity and deadline). The framework presented here is largely model-agnostic and hence is directly applicable to many other types of natural disaster models such as floods, earthquakes, and cyclones. We will expand the work in the future to cover such models. The fault-tolerance of the proposed system will be explored further as well in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank Samuel Ferguson and Rochelle Richards at the Tasmanian Fire Service for providing the data and Spark configurations used as examples within this study. The authors are grateful to ‘‘Big System’’ research team at University of Tasmania for continued support. This work was supported by Data61, Commonwealth Scientific and Industrial Research Organization (CSIRO) and University of Tasmania (Tasmania Graduate Research Scholarship 2018).

Appendix 1. Symbols and Notations Used

Symbols	Description
u_k	User Request
M_i	Worker of flavor type i
p_{M_i}	Number of worker of type M_i
C_{M_i}	Operation cost of worker M_i
t_{j,M_i}	Operation time of worker M_i
N_S	Total number of simulations in user request
n_{s,j,M_i}	Number of simulation run by worker j of type M_i
T_u	User requirement of Time
C_u	User requirement of Cost
N	Total number of workers
$u_{k,c}$	Cost requirement associated with u_k
$u_{k,d}$	Time requirement associated with u_k
A_{M_i}	A number of M_i instances
$t_{k,sys}$	Time for which u_k is in the system
UL_k	Urgency level for u_k
t_{new}	Time required to create new instances
λ	Arrival rate of user requests
μ	Service rate of proposed system

(continued on next page)

(*continued*)

Symbols	Description
D_{type}	Instance type (on-demand or spot)
bid_{price}	Bid amount for spot instances
S_t	Pre-defined urgency standard
Δn	Number of instances required to be created
CP	Capacity Planner
N_{M_i}	Minimum number of M_i instances required in CP
N_c	Minimum Cluster Size in the system decided by CP
Δnd	Number of on-demand instances to be created or deleted
CP_q	Queue in Capacity Planner
Q	Job Queue
R	Queue of requests being served by the system
S_i	i^{th} subjob in a user request
r	Risk metric
C	Operation Cost
X	Total traffic offered in Erlang
$B(X,y)$	Blocking probability for y servers with
X	Traffic offered
t	Time instant
n_1	Number of Simulation in a single process
x	Number of Spark process
$N_{S,x}$	Total number of simulation in x Spark processes
x_n	Number of simulation in each Spark process

References

Ang, T., Ng, W., Ling, T., Por, L., Liew, C., 2009. A bandwidth-aware job grouping-based scheduling on grid environment. Inf. Technol. J. 8 (3), 372–377.

AWS pricing. https://aws.amazon.com/ec2/pricing/on-demand/. (Accessed 12 September 2018).

Behzad, B., Padmanabhan, A., Liu, Y., Liu, Y., Wang, S., 2011. Integrating cybergis gateway with windows azure: a case study on modflow groundwater simulation. In: Proceedings of the ACM SIGSPATIAL Second International Workshop on High Performance and Distributed Geographic Information Systems. ACM, pp. 26–29.

Bicer, T., Chiu, D., Agrawal, G., 2012. Time and cost sensitive data-intensive computing on hybrid clouds. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012). IEEE Computer Society, pp. 636–643.

Candeia, D., Araujo, R., Lopes, R., Brasileiro, F., 2010. Investigating business-driven cloudburst schedulers for e-science bag-of-tasks applications. In: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, pp. 343–350.

Duan, R., Prodan, R., 2014. Cooperative scheduling of bag-of-tasks workflows on hybrid clouds. Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, pp. 439–446.

Eriksson, H., Raciti, M., Basile, M., Cunsolo, A., Fröberg, A., Leifler, O., Ekberg, J., Timpka, T., 2011. A cloud-based simulation architecture for pandemic influenza simulation. In: AMIA Annual Symposium Proceedings, vol 2011. American Medical Informatics Association, p. 364.

Farsite, M. A. Finney, 1998. Fire area simulator-model development and evaluation. In: Res. Pap. RMRS-RP-4, vol 4. US Department of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, UT, p. 47. Revised 2004.

Garg, S., Forbes-Smith, N., Hilton, J., Sparkcloud, M. Prakash, 2018. A cloud-based elastic bushfire simulation service. Rem. Sens. 10 (1), 74.

Huang, H., Wang, L., Tak, B.C., Wang, L., Tang, C., 2013a. Cap3: a cloud auto-provisioning framework for parallel processing using on-demand and spot instances. In: IEEE Sixth International Conference on Cloud Computing. IEEE, pp. 228–235.

Huang, Q., Yang, C., Benedict, K., Chen, S., Rezgui, A., Xie, J., 2013b. Utilize cloud computing to support dust storm forecasting. Int. J. Digital Earth 6 (4), 338–355.

Huang, Q., Li, J., Li, Z., 2018. A geospatial hybrid cloud platform based on multi-sourced computing and model resources for geosciences. Int. J. Digital Earth 11 (12), 1184–1204.

Huntington, J.L., Hegewisch, K.C., Daudert, B., Morton, C.G., Abatzoglou, J.T., McEvoy, D.J., Erickson, T., 2017. Climate engine: cloud computing and visualization of climate and remote sensing data for advanced natural resource monitoring and process understanding. Bull. Am. Meteorol. Soc. 98 (11), 2397–2410.

Kalabokidis, K.D., Gatzojannis, S., Galatsidas, S., 2002. Introducing wildfire into forest management planning: towards a conceptual approach. For. Ecol. Manag. 158 (1–3), 41–50.

Kalabokidis, K., Athanasis, N., Gagliardi, F., Karayiannis, F., Palaiologou, P., Parastatidis, S., Vasilakos, C., 2013. Virtual Fire: a web-based GIS platform for forest fire control. Ecol. Inf. 16, 62–69.

Kalabokidis, K., Athanasis, N., Vasilakos, C., Palaiologou, P., 2014. Porting of a wildfire risk and fire spread application into a cloud computing environment. Int. J. Geogr. Inf. Sci. 28 (3), 541–552.

Keat, N.W., Fong, A.T., Chaw, L.T., Sun, L.C., 2006. Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. Malays. J. Comput. Sci. 19 (2), 117–126.

Li, Z., Yang, C., Huang, Q., Liu, K., Sun, M., Xia, J., 2017. Building model as a service to support geosciences. Comput. Environ. Urban Syst. 61, 141–152.

Mann, Z.Á., 2015. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput. Surv. 48 (1), 11.

Messerli, E.J., 1972. B.S.T.J. brief: Proof of a convexity property of the erlang B formula. Bell Syst. Tech. J. 51 (4), 951–953. https://doi.org/10.1002/j.1538-7305.1972.tb01956.x.

Miller, C., Hilton, J., Sullivan, A., Prakash, M., 2015. Spark–a bushfire spread prediction tool. In: International Symposium on Environmental Software Systems. Springer, pp. 262–271.

Montgomery, K., Mundt, C., 2010. A new paradigm for integrated environmental monitoring. In: Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application. ACM, p. 29.

Muthuvelu, N., Chai, I., Eswaran, C., 2008. An adaptive and parameterized job grouping algorithm for scheduling grid jobs. In: 10th International Conference on Advanced Communication Technology, vol 2. IEEE, pp. 975–980.

Muthuvelu, N., Chai, I., Chikkannan, E., Buyya, R., 2010. On-line task granularity adaptation for dynamic grid applications. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer, pp. 266–277.

Muthuvelu, N., Vecchiola, C., Chai, I., Chikkannan, E., Buyya, R., 2013. Task granularity policies for deploying bag-of-task applications on global grids. Future Generat. Comput. Syst. 29 (1), 170–181.

Nectar cloud. https://nectar.org.au/research-cloud/. (Accessed 12 May 2018).

Pajorová, E., Hluchý, L., 2011. Scientific gateway and visualization tool. In: Computational Intelligence in Security for Information Systems. Springer, pp. 246–250.

Thai, L., Varghese, B., Barker, A., 2018. A survey and taxonomy of resource optimisation for executing bag-of-task applications on public clouds. Future Generat. Comput. Syst. 82, 1–11.

Tijms, H.C., Van Hoorn, M.H., Federgruen, A., 1981. Approximations for the steady-state probabilities in the m/g/c queue. Adv. Appl. Probab. 13 (1), 186–206.

Ujjwal, K., Garg, S., Hilton, J., Aryal, J., Forbes-Smith, N., 2019. Cloud computing in natural hazard modeling systems: current research trends and future directions. Int. J. Disas. Risk Reduct. 101188.

Varghese, B., Buyya, R., 2018. Next generation cloud computing: new trends and research directions. Future Generat. Comput. Syst. 79, 849–861.

Wan, Z., Hong, Y., Khan, S., Gourley, J., Flamig, Z., Kirschbaum, D., Tang, G., 2014. A cloud-based global flood disaster community cyber-infrastructure: development and demonstration. Environ. Model. Software 58, 86–94.